

# 3806ICT - Assignment 1

Nick van der Merwe - s5151332 - nick.vandermmerwe@griffithuni.edu.au

May 3, 2021

## 1 Introduction

In this piece a controller is defined for a robot using a proportional integral derivative controller. This is split into two tasks, defining the PID algorithm and the Kalman filter in the assignment. However, the architecture of the solution will contain four parts: the sonar wrapper, the PID service, the Kalman filter service and the controller. Essentially, the sonar wrapper will turn the sonar topic into a service so that the controller does not subscribe to it. The PID and the Kalman filter will be purely functions without any storage so that they can be reused in the future. Lastly, the controller node will manage all of the data.

This report is organised based on the content in the nodes. Each section will introduce the necessary theory and explain how that is converted into code. With that in mind, Section 2 contains the sonar wrapper, Section 3 covers the PID equation (not defining the constants), Section 4 runs over the kalman filter (not finding the variance), and Section 5 defines the controller. The controller contains how all the previous nodes are tied together, including how the constants in PID are defined and variance is found for the Kalman filter.

## 2 Sonar Wrapper Node

This is quite straight forwards as it only requires us to save the last published value in a variable. As a design decision to avoid global variables it was made as a class. The code is visible in Figure 1a and 1b.

```
1 ---
2 assignment1_setup/Sonars_readings
```

(a)

```
1 //
2 // sonarwrapper.cpp
3 // Copyright (C) 2021 Nick van der Merwe - nick.vandermmerwe@griffithuni.edu.au
4 //
5 // The purpose of this is to wrap the sonars topic and return
6 // what it says
7 //
8 #include "ros/ros.h"
9 #include "assignment1_setup/Sonars.h"
10 #include "assignment1_getSonarReadings.h"
11 #include <iomanip>
12
13 class SonarReader{
14 public:
15     SonarReader() = default;
16
17     void subCallback(const assignment1_setup::Sonars &msg){
18         lastReading = msg;
19     }
20
21     bool getSonarReadings(
22         assignment1_getSonarReadings::Request &req,
23         assignment1_getSonarReadings::Response &res){
24         res.readings = lastReading;
25         ROS_INFO("sending %d", (unsigned int)res.readings.distance);
26         return true;
27     }
28
29     private:
30         assignment1_setup::Sonars lastReading;
31 };
32
33 int main(int argc, char **argv){
34     ros::init(argc, argv, "sonar_wrapper_node");
35     ros::NodeHandle nodeHandle;
36
37     SonarReader sr{};
38     ros::Subscriber sub =
39         nodeHandle.subscribe("sonars", 1000,
40                               &SonarReader::subCallback, &sr);
41
42     ros::ServiceServer getSonarReadings =
43         nodeHandle.advertiseService("sonar_wrapper",
44                                     &SonarReader::getSonarReadings, &sr);
45
46     ROS_INFO("Ready to manage requests");
47
48     ros::spin();
49     return EXIT_SUCCESS;
50 }
```

(b)

Figure 1: Sonar wrapper srv in a and code in b

### 3 Task 1: PID Node

Before the coding section, the mathematics behind what a proportional integral derivative controller does must be explained. In short, the goal of the controller is to minimise the error that is reported. As stated in the task, the formula to be used here is in Equation 1.

$$\begin{aligned} Y(t) &= K_p e(t) + K_d \frac{de(t)}{dt} + K_i \int_0^t e(t') dt' \\ &= P(t) + D(t) + I(t) \end{aligned} \quad (1)$$

Where  $e(t)$  is the input error: in our case the distance from the bowl.  $t$  is the time,  $K_p$  is proportional gain,  $K_d$  is the derivative gain and  $K_i$  is the integral gain.

To make elaborating how this equation functions simpler, the first thing to do is to consider each of the terms separately as they each represent a part of **Proportional Integral Derivative**. To summarise, the proportional section represents the current error, the integral is the error in the past and the derivative is a prediction of the error into the future [M., 2000]. The logic behind this is that for the P term its just the current error, the integral is the area under the curve in the past and the derivative is the change at that specific point. These are each adjusted to the environment by using the K terms on each term.

Now the next issue is that this representation uses continuous time whereas in computational mathematics everything must be discrete - in this case our sonar only reports every 10ms. The exact maths for converting this into discrete values was covered in the lab content so it will not be copied over here, but rather explained.

The first step is to define time as discrete by making  $T = 10ms$  and  $t_i = T + t_i - 1$ . Technically this would be all that's necessary, but the exact way to calculate the derivative and integral is not known. For these the conceptual meanings can be used: a derivative is the change in the output of a function in the smallest unit of time and an integral is the area under the curve from one time to another. Another fact is that technically a derivative forwards (current minus future) is the same as backwards (current minus last value) in continuous time due to it being infinitely small.

To take this and define them discretely with logic is simple. A derivative is its current value subtracted by its last value, all divided by T (since that is the smallest unit of time we have). Next the integral is simply the sum of all the previous error values times T. This can conceptually be seen as how the area of a square is found at a low level without multiplication: split it into strips of size 1 (in our case T) and add these areas together. Including the K terms, our equations are in Equations 2, 3 and 4.

$$P(t) = K_p e(t) \quad (2)$$

$$D(t) = K_d \frac{e(t_i) - e(t_{i-1})}{T} \quad (3)$$

$$I(t_i) = K_i \sum_{i=1}^i T e(t_i) \quad (4)$$

Programmatically, it would be inefficient to do the integral by summing up everything in this manner every time it is calculated, so realistically the value is simply returned on its own in the function so that it can just be added to once in the next run. This is represented as Equation 5.

$$\begin{aligned} I(t_i) &= K_i f(t) \\ f(t) &= e(t_i)T + f(t_{i-1}) \end{aligned} \quad (5)$$

The PID srv file and C++ code is visible in figure 2.

```

1 float64 error
2 float64 lastError
3 float64 totalFValue
4 float64 K_p
5 float64 K_i
6 float64 K_d
7 float64 T
8 ---
9 float64 y
10 float64 lastError
11 float64 totalFValue

```

```

1 /*
2  * Written by Nick van der Merwe - s5151332
3  */
4 #include "ros/ros.h"
5 #include "assignment1_setup/Sonars.h"
6 #include "assignment1/pid_algorithm.h"
7 #include "geometry_msgs/Twist.h"
8 #include <cmath>
9 #include <algorithm>
10
11 /*
12  * We want to calculate the PID. To increase readability we
13  * are going to split each of these formulas into a separate function:
14  *  $y(t_i) = P(t_i) + I(t_i) + D(t_i)$ 
15  *  $P(t_i) = K_p * e(t_i)$ 
16  *  $I(t_i) = e(t_i) + I(t_{i-1})$ 
17  *  $D(t_i) = K_d * (e(t_i) - e(t_{i-1})) / T$ 
18  *  $e(t) = \text{distance from object at } t$ 
19  *  $T = \text{latency}$ 
20  */
21
22 bool pidAlgorithm(
23     assignment1::pid_algorithm::Request & req,
24     assignment1::pid_algorithm::Response & res){
25     double P = req.K_p * req.error;
26     res.totalFValue = req.error * req.T + req.totalFValue;
27     double I = req.K_i * res.totalFValue;
28     // split D into top and bottom so its more readable
29     double && topD = req.error - req.lastError;
30     double D = req.K_d * (topD / req.T);
31
32     res.y = P + I + D;
33     // checking the documentation 22.0 is the maximum speed
34     res.y = std::min(res.y, 0.22);
35     res.y = std::max(res.y, 0.0);
36
37     ROS_INFO("lastError: %u totalFValue %lf",
38             (unsigned int) res.lastError, res.totalFValue);
39     ROS_INFO("K_p: %lf K_i %lf K_d %lf", req.K_p, req.K_i, req.K_d);
40     ROS_INFO("P: %lf, I: %lf, D: %lf", P, I, D);
41     ROS_INFO("Returning y as %lf", res.y);
42     return true;
43 }
44
45 int main(int argc, char **argv){
46     ros::init(argc, argv, "pid_algorithm_node");
47     ros::NodeHandle nodeHandle;
48
49     ros::ServiceServer getSonarReadings =
50         nodeHandle.advertiseService("pid_algorithm", pidAlgorithm);
51
52     ROS_INFO("Ready to manage requests");
53     ros::spin();
54
55     return EXIT_SUCCESS;
56 }

```

Figure 2: The PID srv and cpp files respectively

## 4 Task 2: Kalman Filter Node

The goal of a Kalman filter is to get rid of noise in sonar readings through maths, and in this case odometry readings. The equations to be used are defined in Equations 6.

$$\begin{aligned} K_i &= \frac{P_i^-}{P_i^- + R_i} \\ y_i &= y_i^- + K_i(z_i - y_i^-) \\ P_i &= (1 - K_i)P_i^- \end{aligned} \tag{6}$$

Where  $z_i$  is the sonar's reading,  $y_i^-$  is the estimation of how far the robot is from the bowl at step  $i$ ,  $P_i^-$  is the predicted variance at  $i$ , and  $R_i$  is the initial variance read. Since there are no continuous variables in this, there is no need for this to be changed such as the PID. However, each of these should still be properly explained.

The naive way to solve the issue of a normally distributed noisy sonar would simply be to measure the median every time. However, to do that the robot would have to sit there for several seconds, move forwards slightly then sit there and measure the median again. Instead, what the Kalman filter aims to do is take the measuring once, move, (in our case) consider how far it moved through the odometry, apply the Kalman filter to this and get an estimate of how far it is [Siegward et al., 2011]. The reason why the odometry alone would not be enough is visible in how the sonar functions. Consider the case where the bowl lies 13 degrees to the right of the way the robot is facing. If we only use the euclidean distance measured by the odometry system, it would not actually be the distance from the bowl. From all of this, we can see the need for why the Kalman filter is necessary.

For now the focus is on applying the Kalman filter step out of the steps detailed previously which is all in equation 6. In short, the goal of this step is to give readings varying weight depending on the certainty of them. This weight is given by the Kalman gain by taking the predicted variance divided by itself added to the overall variance. Then we can add the difference in the reading and the estimated position to its estimated position to find  $y_i$ . To calculate the next predicted variance is typically complicated, but was compressed into multiplying  $P_i^-$  by the complement of the Kalman gain  $(1 - K_i)$ . In practice this just makes later readings have less weight.

In an attempt to understand this particular equation, an interesting proof by induction was found that makes calculating the variance irrelevant. Essentially it's possible to remove  $P_i$ ,  $P_i^-$  and  $R_i$  from the equations. First, let's change the notation so that the pattern is more obvious.

$$\begin{aligned} K_i &= \frac{P_i}{P_i + R} \\ P_{i+1} &= (1 - K_i)P_i \\ P_0 &= R \end{aligned}$$

First we define our base case of  $K_0$

$$\begin{aligned} K_0 &= \frac{P_0}{P_0 + R} \\ P_0 &= R \\ K_0 &= \frac{P_0}{2P_0} \\ K_0 &= \frac{1}{2} \end{aligned}$$

Now in induction we just have to prove the  $i + 1$  case

$$\begin{aligned}
 P_{i+1} &= R(1 - K_i) \\
 K_{i+1} &= \frac{P_{i+1}}{P_{i+1} + R} \\
 &= \frac{R(1 - K_i)}{R(1 - K_i) + R} \\
 &= \frac{R(1 - K_i)}{R - RK_i + R} \\
 &= \frac{R(1 - K_i)}{2R - RK_i} \\
 &= \frac{R(1 - K_i)}{R(2 - K_i)} \\
 &= \frac{1 - K_i}{2 - K_i}
 \end{aligned}$$

So essentially we found that

$$\begin{aligned}
 K_0 &= \frac{1}{2} \\
 K_{i+1} &= \frac{1 - K_i}{2 - K_i}
 \end{aligned}$$

To elaborate, the reasons why this is happening is because  $P_i$  does not change in the middle between calculations as the odometry has no variance. If it had a changing variance and  $P_i$  were to suddenly spike, it would give less weight to the odometry and more weight to the sonar reader. Essentially it's throwing the sonars out the window by decreasing the value of the Kalman gain quickly. However, it still means that the entire variance step could be skipped as a result.

With this being said, it will just be ignored. It's an interesting proof that makes variance useless, but taking it out will probably remove a section of the assignment that the assignment writer intended the student to do. It's essentially just evidence of extensive research around understanding the formulas at work.

Moving onto the implementation, its a service that uses the exact formulas on the task sheet visible in figure 3.

```

1 float64 R_i
2 uint16 z_i
3 float64 y_i_estimate
4 float64 P_i_estimate
5 ---
6 float64 y_i
7 float64 P_i

```

(a)

```

1 //
2 // * By Nick van der Merwe - s5151332
3 //
4 #include <iomanip>
5 #include <gazebo_msgs/ModelStates.h>
6 #include "ros/ros.h"
7 #include "assignment1/kalman_filter.h"
8
9 ...
10
11 bool kalmanFilter(
12     assignment1::kalman_filter::Request &req,
13     assignment1::kalman_filter::Response &res) {
14     double K_i = req.P_i_estimate / (req.P_i_estimate + req.R_i);
15     res.y_i = req.y_i_estimate + K_i * (req.z_i - req.y_i_estimate);
16     ROS_INFO("P_i_estimate (before) %lf", req.P_i_estimate);
17     req.P_i_estimate = (1 - K_i) * req.P_i_estimate;
18     ROS_INFO("K_i %lf R_i %lf z_i %lf y_i_estimate %lf P_i_estimate(after) %lf",
19             K_i, req.R_i, (unsigned int)req.z_i, req.y_i_estimate, req.P_i_estimate);
20
21     ROS_INFO("Returning y_i as %lf", res.y_i);
22     return true;
23 }
24
25 int main(int argc, char **argv) {
26     ros::init(argc, argv, "kalman_filter_node");
27     ros::NodeHandle nodeHandle;
28
29     ros::ServiceServer findFilteredOutput =
30         nodeHandle.advertiseService("kalman_filter", kalmanFilter);
31
32     ROS_INFO("Ready to manage requests");
33     ros::spin();
34
35     return EXIT_SUCCESS;
36 }

```

(b)

Figure 3: Kalman filter implementation, srv file in 3a and code in 3b

## 5 Controller Node

### 5.1 PID constants

As picking the PID terms is not an essential step and, only needs to be justified, we can use manual selection. To define our PID terms we must consider the goal of how we wish the robot to move. A reasonably logical approach would be to make the robot run at maximum speed until it comes close to the bowl and then slow down when it's closer so that it can accurately move. To align this with a real world goal, it's wanting to drive to a location as fast as possible yet be safe. In other words, when a car is parking it slows down so it can move more accurately.

The  $K_p$  term does exactly this by changing its speed in respect to its distance from the target. As such we can pick a point that looks reasonably close (see figure 4) and define that as the point it should start slowing. To work out the  $K_p$  from this all that must be done is the maximum speed should be found when it's at 100 units by doing  $y(t) = K_p e(t) \Rightarrow 0.22 = K_p 100 \Rightarrow K_p = 0.22/100$ .

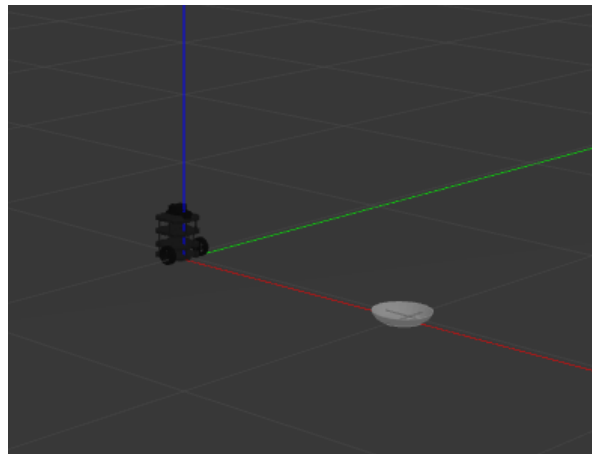


Figure 4: How far the robot looks when it's 100 units away

One issue to consider is that typically the error in PID is not just distance (i.e. its the difference between the goal speed and current speed), which is especially impactful to the integral term. Usually when the error hits zero, the integral value is set as a constant - otherwise it would continue to rise as the controller is running [Siegward et al., 2011]. In this case it only occurs when the robot actually reaches the bowl - which would be the end of the problem. In the case that the robot is as far away as its sonar readers allow (around 60,000), it would likely mean that it will just run at maximum speed all the way to the bowl. This results in three options: 1) Set  $K_i$  to close to 0, rendering the entire term next to useless 2) Let the robot just run at maximum speed when its far away 3) Decide on a point when the integral term should stop changing.

To be true to the goal, the best option here would be option 1. Option 2 is in direct conflict with the goal of slowing down when close as it renders it impossible in some situations, and picking an arbitrary point where the integral stops growing would only cause the robot to have a minimum constant speed after a certain point. While  $K_i$  could be set close to zero such that it has a 'conceptual' effect, it's easier to just set it to zero as it would affect the speed in the same way in this environment.

As the derivative term allows the robot to continue going a certain speed or slow down less than usual, it could be useful in the Kalman filter. This is because occasionally the error is filtered to be a negative value which in that case the P term completely stops, and often far too early. Furthermore, even with just the P term on its own in a non-noisy sonar it starts going abysmally slow as it approaches it. As a solution,  $K_d$  is defined such that the robot only lose 80% of its speed every second. To do this, the meaning of what the D term should be understood fully. It takes in the current distance, subtracts the last distance and divides by T. The subtraction operation, in theory, should always lead to a negative number as we are approaching the goal and dividing by T (0.01 seconds) says what the velocity would be for a second - as a negative. From this all that must be done to make sure 20% of the previous velocity is added,  $K_d$  should equal  $-0.2/100$  as

to cancel the negative and represent its velocity in metres (that is the divided by 100) every second.

In reality though, due to the factor that the sonar readings are integers this does not function as intended. The maximum speed is 22 centimetres per second, and its reporting that 100 times every second. In other words, every four (or so) readings the difference would be 1cm, and not every reading has a difference of 0.22. As a result, the D term simply adds four times more than the intended value every four iterations. Since this is a sonar based issue, there is not much of a work around other than turning the sonar reading into a float64, ignoring the issue or setting  $K_d$  to zero. Out of these, the issue was chosen to be ignored as in the Kalman filter this is not what occurs: the readings are floats.

To tie it together, it was decided that  $K_p = 0.22/100$ ,  $K_i = 0$ ,  $K_d = -0.2$ .  $K_p$  makes the robot slow down as it approaches and  $K_d$  prevents the robot from slowing down too abruptly. The issue with  $K_i$  is that it makes the robot either stay at a certain speed, or stay at maximum speed depending on how its implemented.

## 5.2 Calculating Variance

To do this is not any different from usual:

$$\begin{aligned} \text{mean} &= \frac{1}{n} \sum_{i=0}^n x_i \\ \text{variance} &= \frac{\sum_{i=0}^n (x_i - \text{mean})^2}{n - 1} \end{aligned} \tag{7}$$

where  $x_i$  is a collected sample and  $n$  is the number of samples [Bhandari, 2020]. Note that the sample formula is used instead of the population formula. With regards to the number of samples collected, it was decided that 1000 will be used. Realistically, this is hard to justify without other variables involved but at 1000 points it is expected to give ten points for every percentile which should be reasonable.

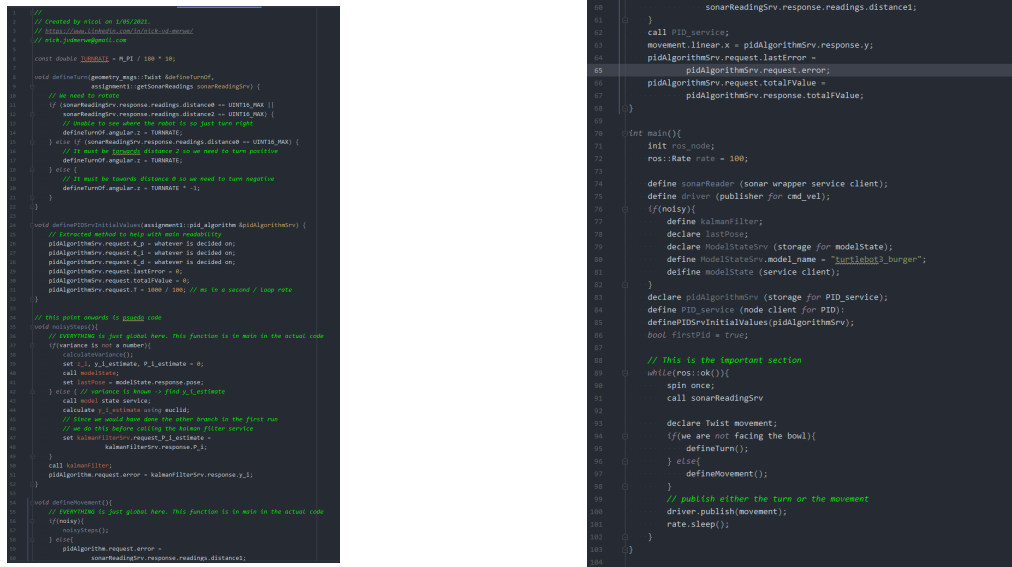
One option is to hard-code in the variance because in the *sonars.cc* source code provided `std :: normal_distribution()` is used and the standard deviation is defined as 100. Meaning, the variance is hard-coded to  $100^2 = 10,000$ . However, hard-coding in the answer would destroy an intended part of the assignment and reduce reusability in the case that the sonar reader's distribution changes. Furthermore, this can be used as justification for why the variance is calculated every run: to improve reusability.

## 5.3 Random turns due to noise

Due to the sonars being capable of randomly shooting to uint16's max (even if the chance is low), it is possible for the robot to accidentally shift out of sight of the bowl if one comes through. To counteract this, it was implemented that the robot should read that it lost sight of the bowl five consecutive times before it turns. While technically possible, the chances of this occurring is low enough that it does not happen.

## 5.4 Structure

A core factor that makes implementing this difficult is that extracting functions from main in ROS is often prohibited due to a couple of methods only being permitted to be called inside main. So for easier understanding, figure 5 was made to grasp the implementation better. However, it should only be used as a guide to the general structure - some extra things were added in the code that was detailed in the justifications, but not the pseudo-code.



```

// Pseudo-code for ROS node
// Includes
#include <iomanip>
#include <vector>
#include <string>
#include <math>
#include "assignment1/setupSensors.h"
#include "assignment1/getSonarReadings.h"
#include "assignment1/pid_algorithm.h"
#include "assignment1/kalman_filter.h"
#include "geometry_msgs/GetModelState.h"
#include "geometry_msgs/Twist.h"
#include "geometry_msgs/Pose.h"
#include <vector>

// Constants
// Maximum is 1.6 radians / second -> half a radian
// One degree = 180/PI
const double TURNRATE = M_PI / 180 * 50;
const int NOISY_SONAR = 1000;
const int NOISY_PID = 1000;

// Main function
int main(int argc, char** argv) {
    // Initialize ROS
    ros::init(argc, argv, "controller");
    ros::NodeHandle nh;

    // Create publishers and subscribers
    ros::Publisher pub_twist = nh.advertise<Twist>("cmd_vel", 1000);
    ros::Subscriber sub_pose = nh.subscribe<Pose>("pose", 10, &poseCallback);

    // Initialize variables
    Pose p1, p2;
    Twist twist;
    double distance;
    double variance;
    double x, y;
    double x1, y1;
    double x2, y2;
    double x3, y3;
    double x4, y4;
    double x5, y5;
    double x6, y6;
    double x7, y7;
    double x8, y8;
    double x9, y9;
    double x10, y10;
    double x11, y11;
    double x12, y12;
    double x13, y13;
    double x14, y14;
    double x15, y15;
    double x16, y16;
    double x17, y17;
    double x18, y18;
    double x19, y19;
    double x20, y20;
    double x21, y21;
    double x22, y22;
    double x23, y23;
    double x24, y24;
    double x25, y25;
    double x26, y26;
    double x27, y27;
    double x28, y28;
    double x29, y29;
    double x30, y30;
    double x31, y31;
    double x32, y32;
    double x33, y33;
    double x34, y34;
    double x35, y35;
    double x36, y36;
    double x37, y37;
    double x38, y38;
    double x39, y39;
    double x40, y40;
    double x41, y41;
    double x42, y42;
    double x43, y43;
    double x44, y44;
    double x45, y45;
    double x46, y46;
    double x47, y47;
    double x48, y48;
    double x49, y49;
    double x50, y50;
    double x51, y51;
    double x52, y52;
    double x53, y53;
    double x54, y54;
    double x55, y55;
    double x56, y56;
    double x57, y57;
    double x58, y58;
    double x59, y59;
    double x60, y60;
    double x61, y61;
    double x62, y62;
    double x63, y63;
    double x64, y64;
    double x65, y65;
    double x66, y66;
    double x67, y67;
    double x68, y68;
    double x69, y69;
    double x70, y70;
    double x71, y71;
    double x72, y72;
    double x73, y73;
    double x74, y74;
    double x75, y75;
    double x76, y76;
    double x77, y77;
    double x78, y78;
    double x79, y79;
    double x80, y80;
    double x81, y81;
    double x82, y82;
    double x83, y83;
    double x84, y84;
    double x85, y85;
    double x86, y86;
    double x87, y87;
    double x88, y88;
    double x89, y89;
    double x90, y90;
    double x91, y91;
    double x92, y92;
    double x93, y93;
    double x94, y94;
    double x95, y95;
    double x96, y96;
    double x97, y97;
    double x98, y98;
    double x99, y99;
    double x100, y100;
    double x101, y101;
    double x102, y102;
    double x103, y103;
    double x104, y104;
    double x105, y105;
    double x106, y106;
    double x107, y107;
    double x108, y108;
    double x109, y109;
    double x110, y110;
    double x111, y111;
    double x112, y112;
    double x113, y113;
    double x114, y114;
    double x115, y115;
    double x116, y116;
    double x117, y117;
    double x118, y118;
    double x119, y119;
    double x120, y120;
    double x121, y121;
    double x122, y122;
    double x123, y123;
    double x124, y124;
    double x125, y125;
    double x126, y126;
    double x127, y127;
    double x128, y128;
    double x129, y129;
    double x130, y130;
    double x131, y131;
    double x132, y132;
    double x133, y133;
    double x134, y134;
    double x135, y135;
    double x136, y136;
    double x137, y137;
    double x138, y138;
    double x139, y139;
    double x140, y140;
    double x141, y141;
    double x142, y142;
    double x143, y143;
    double x144, y144;
    double x145, y145;
    double x146, y146;
    double x147, y147;
    double x148, y148;
    double x149, y149;
    double x150, y150;
    double x151, y151;
    double x152, y152;
    double x153, y153;
    double x154, y154;
    double x155, y155;
    double x156, y156;
    double x157, y157;
    double x158, y158;
    double x159, y159;
    double x160, y160;
    double x161, y161;
    double x162, y162;
    double x163, y163;
    double x164, y164;
    double x165, y165;
    double x166, y166;
    double x167, y167;
    double x168, y168;
    double x169, y169;
    double x170, y170;
    double x171, y171;
    double x172, y172;
    double x173, y173;
    double x174, y174;
    double x175, y175;
    double x176, y176;
    double x177, y177;
    double x178, y178;
    double x179, y179;
    double x180, y180;
    double x181, y181;
    double x182, y182;
    double x183, y183;
    double x184, y184;
    double x185, y185;
    double x186, y186;
    double x187, y187;
    double x188, y188;
    double x189, y189;
    double x190, y190;
    double x191, y191;
    double x192, y192;
    double x193, y193;
    double x194, y194;
    double x195, y195;
    double x196, y196;
    double x197, y197;
    double x198, y198;
    double x199, y199;
    double x200, y200;
    double x201, y201;
    double x202, y202;
    double x203, y203;
    double x204, y204;
    double x205, y205;
    double x206, y206;
    double x207, y207;
    double x208, y208;
    double x209, y209;
    double x210, y210;
    double x211, y211;
    double x212, y212;
    double x213, y213;
    double x214, y214;
    double x215, y215;
    double x216, y216;
    double x217, y217;
    double x218, y218;
    double x219, y219;
    double x220, y220;
    double x221, y221;
    double x222, y222;
    double x223, y223;
    double x224, y224;
    double x225, y225;
    double x226, y226;
    double x227, y227;
    double x228, y228;
    double x229, y229;
    double x230, y230;
    double x231, y231;
    double x232, y232;
    double x233, y233;
    double x234, y234;
    double x235, y235;
    double x236, y236;
    double x237, y237;
    double x238, y238;
    double x239, y239;
    double x240, y240;
    double x241, y241;
    double x242, y242;
    double x243, y243;
    double x244, y244;
    double x245, y245;
    double x246, y246;
    double x247, y247;
    double x248, y248;
    double x249, y249;
    double x250, y250;
    double x251, y251;
    double x252, y252;
    double x253, y253;
    double x254, y254;
    double x255, y255;
    double x256, y256;
    double x257, y257;
    double x258, y258;
    double x259, y259;
    double x260, y260;
    double x261, y261;
    double x262, y262;
    double x263, y263;
    double x264, y264;
    double x265, y265;
    double x266, y266;
    double x267, y267;
    double x268, y268;
    double x269, y269;
    double x270, y270;
    double x271, y271;
    double x272, y272;
    double x273, y273;
    double x274, y274;
    double x275, y275;
    double x276, y276;
    double x277, y277;
    double x278, y278;
    double x279, y279;
    double x280, y280;
    double x281, y281;
    double x282, y282;
    double x283, y283;
    double x284, y284;
    double x285, y285;
    double x286, y286;
    double x287, y287;
    double x288, y288;
    double x289, y289;
    double x290, y290;
    double x291, y291;
    double x292, y292;
    double x293, y293;
    double x294, y294;
    double x295, y295;
    double x296, y296;
    double x297, y297;
    double x298, y298;
    double x299, y299;
    double x300, y300;
    double x301, y301;
    double x302, y302;
    double x303, y303;
    double x304, y304;
    double x305, y305;
    double x306, y306;
    double x307, y307;
    double x308, y308;
    double x309, y309;
    double x310, y310;
    double x311, y311;
    double x312, y312;
    double x313, y313;
    double x314, y314;
    double x315, y315;
    double x316, y316;
    double x317, y317;
    double x318, y318;
    double x319, y319;
    double x320, y320;
    double x321, y321;
    double x322, y322;
    double x323, y323;
    double x324, y324;
    double x325, y325;
    double x326, y326;
    double x327, y327;
    double x328, y328;
    double x329, y329;
    double x330, y330;
    double x331, y331;
    double x332, y332;
    double x333, y333;
    double x334, y334;
    double x335, y335;
    double x336, y336;
    double x337, y337;
    double x338, y338;
    double x339, y339;
    double x340, y340;
    double x341, y341;
    double x342, y342;
    double x343, y343;
    double x344, y344;
    double x345, y345;
    double x346, y346;
    double x347, y347;
    double x348, y348;
    double x349, y349;
    double x350, y350;
    double x351, y351;
    double x352, y352;
    double x353, y353;
    double x354, y354;
    double x355, y355;
    double x356, y356;
    double x357, y357;
    double x358, y358;
    double x359, y359;
    double x360, y360;
    double x361, y361;
    double x362, y362;
    double x363, y363;
    double x364, y364;
    double x365, y365;
    double x366, y366;
    double x367, y367;
    double x368, y368;
    double x369, y369;
    double x370, y370;
    double x371, y371;
    double x372, y372;
    double x373, y373;
    double x374, y374;
    double x375, y375;
    double x376, y376;
    double x377, y377;
    double x378, y378;
    double x379, y379;
    double x380, y380;
    double x381, y381;
    double x382, y382;
    double x383, y383;
    double x384, y384;
    double x385, y385;
    double x386, y386;
    double x387, y387;
    double x388, y388;
    double x389, y389;
    double x390, y390;
    double x391, y391;
    double x392, y392;
    double x393, y393;
    double x394, y394;
    double x395, y395;
    double x396, y396;
    double x397, y397;
    double x398, y398;
    double x399, y399;
    double x400, y400;
    double x401, y401;
    double x402, y402;
    double x403, y403;
    double x404, y404;
    double x405, y405;
    double x406, y406;
    double x407, y407;
    double x408, y408;
    double x409, y409;
    double x410, y410;
    double x411, y411;
    double x412, y412;
    double x413, y413;
    double x414, y414;
    double x415, y415;
    double x416, y416;
    double x417, y417;
    double x418, y418;
    double x419, y419;
    double x420, y420;
    double x421, y421;
    double x422, y422;
    double x423, y423;
    double x424, y424;
    double x425, y425;
    double x426, y426;
    double x427, y427;
    double x428, y428;
    double x429, y429;
    double x430, y430;
    double x431, y431;
    double x432, y432;
    double x433, y433;
    double x434, y434;
    double x435, y435;
    double x436, y436;
    double x437, y437;
    double x438, y438;
    double x439, y439;
    double x440, y440;
    double x441, y441;
    double x442, y442;
    double x443, y443;
    double x444, y444;
    double x445, y445;
    double x446, y446;
    double x447, y447;
    double x448, y448;
    double x449, y449;
    double x450, y450;
    double x451, y451;
    double x452, y452;
    double x453, y453;
    double x454, y454;
    double x455, y455;
    double x456, y456;
    double x457, y457;
    double x458, y458;
    double x459, y459;
    double x460, y460;
    double x461, y461;
    double x462, y462;
    double x463, y463;
    double x464, y464;
    double x465, y465;
    double x466, y466;
    double x467, y467;
    double x468, y468;
    double x469, y469;
    double x470, y470;
    double x471, y471;
    double x472, y472;
    double x473, y473;
    double x474, y474;
    double x475, y475;
    double x476, y476;
    double x477, y477;
    double x478, y478;
    double x479, y479;
    double x480, y480;
    double x481, y481;
    double x482, y482;
    double x483, y483;
    double x484, y484;
    double x485, y485;
    double x486, y486;
    double x487, y487;
    double x488, y488;
    double x489, y489;
    double x490, y490;
    double x491, y491;
    double x492, y492;
    double x493, y493;
    double x494, y494;
    double x495, y495;
    double x496, y496;
    double x497, y497;
    double x498, y498;
    double x499, y499;
    double x500, y500;
    double x501, y501;
    double x502, y502;
    double x503, y503;
    double x504, y504;
    double x505, y505;
    double x506, y506;
    double x507, y507;
    double x508, y508;
    double x509, y509;
    double x510, y510;
    double x511, y511;
    double x512, y512;
    double x513, y513;
    double x514, y514;
    double x515, y515;
    double x516, y516;
    double x517, y517;
    double x518, y518;
    double x519, y519;
    double x520, y520;
    double x521, y521;
    double x522, y522;
    double x523, y523;
    double x524, y524;
    double x525, y525;
    double x526, y526;
    double x527, y527;
    double x528, y528;
    double x529, y529;
    double x530, y530;
    double x531, y531;
    double x532, y532;
    double x533, y533;
    double x534, y534;
    double x535, y535;
    double x536, y536;
    double x537, y537;
    double x538, y538;
    double x539, y539;
    double x540, y540;
    double x541, y541;
    double x542, y542;
    double x543, y543;
    double x544, y544;
    double x545, y545;
    double x546, y546;
    double x547, y547;
    double x548, y548;
    double x549, y549;
    double x550, y550;
    double x551, y551;
    double x552, y552;
    double x553, y553;
    double x554, y554;
    double x555, y555;
    double x556, y556;
    double x557, y557;
    double x558, y558;
    double x559, y559;
    double x560, y560;
    double x561, y561;
    double x562, y562;
    double x563, y563;
    double x564, y564;
    double x565, y565;
    double x566, y566;
    double x567, y567;
    double x568, y568;
    double x569, y569;
    double x570, y570;
    double x571, y571;
    double x572, y572;
    double x573, y573;
    double x574, y574;
    double x575, y575;
    double x576, y576;
    double x577, y577;
    double x578, y578;
    double x579, y579;
    double x580, y580;
    double x581, y581;
    double x582, y582;
    double x583, y583;
    double x584, y584;
    double x585, y585;
    double x586, y586;
    double x587, y587;
    double x588, y588;
    double x589, y589;
    double x590, y590;
    double x591, y591;
    double x592, y592;
    double x593, y593;
    double x594, y594;
    double x595, y595;
    double x596, y596;
    double x597, y597;
    double x598, y598;
    double x599, y599;
    double x600, y600;
    double x601, y601;
    double x602, y602;
    double x603, y603;
    double x604, y604;
    double x605, y605;
    double x606, y606;
    double x607, y607;
    double x608, y608;
    double x609, y609;
    double x610, y610;
    double x611, y611;
    double x612, y612;
    double x613, y613;
    double x614, y614;
    double x615, y615;
    double x616, y616;
    double x617, y617;
    double x618, y618;
    double x619, y619;
    double x620, y620;
    double x621, y621;
    double x622, y622;
    double x623, y623;
    double x624, y624;
    double x625, y625;
    double x626, y626;
    double x627, y627;
    double x628, y628;
    double x629, y629;
    double x630, y630;
    double x631, y631;
    double x632, y632;
    double x633, y633;
    double x634, y634;
    double x635, y635;
    double x636, y636;
    double x637, y637;
    double x638, y638;
    double x639, y639;
    double x640, y640;
    double x641, y641;
    double x642, y642;
    double x643, y643;
    double x644, y644;
    double x645, y645;
    double x646, y646;
    double x647, y647;
    double x648, y648;
    double x649, y649;
    double x650, y650;
    double x651, y651;
    double x652, y652;
    double x653, y653;
    double x654, y654;
    double x655, y655;
    double x656, y656;
    double x657, y657;
    double x658, y658;
    double x659, y659;
    double x660, y660;
    double x661, y661;
    double x662, y662;
    double x663, y663;
    double x664, y664;
    double x665, y665;
    double x666, y666;
    double x667, y667;
    double x668, y668;
    double x669, y669;
    double x670, y670;
    double x671, y671;
    double x672, y672;
    double x673, y673;
    double x674, y674;
    double x675, y675;
    double x676, y676;
    double x677, y677;
    double x678, y678;
    double x679, y679;
    double x680, y680;
    double x681, y681;
    double x682, y682;
    double x683, y683;
    double x684, y684;
    double x685, y685;
    double x686, y686;
    double x687, y687;
    double x688, y688;
    double x689, y689;
    double x690, y690;
    double x691, y691;
    double x692, y692;
    double x693, y693;
    double x694, y694;
    double x695, y695;
    double x696, y696;
    double x697, y697;
    double x698, y698;
    double x699, y699;
    double x700, y700;
    double x701, y701;
    double x702, y702;
    double x703, y703;
    double x704, y704;
    double x705, y705;
    double x706, y706;
    double x707, y707;
    double x708, y708;
    double x709, y709;
    double x710, y710;
    double x711, y711;
    double x712, y712;
    double x713, y713;
    double x714, y714;
    double x715, y715;
    double x716, y716;
    double x717, y717;
    double x718, y718;
    double x719, y719;
    double x720, y720;
    double x721, y721;
    double x722, y722;
    double x723, y723;
    double x724, y724;
    double x725, y725;
    double x726, y726;
    double x727, y727;
    double x728, y728;
    double x729, y729;
    double x730, y730;
    double x731, y731;
    double x732, y732;
    double x733, y733;
    double x734, y734;
    double x735, y735;
    double x736, y736;
    double x737, y737;
    double x738, y738;
    double x739, y739;
    double x740, y740;
    double x741, y741;
    double x742, y742;
    double x743, y743;
    double x744, y744;
    double x745, y745;
    double x746, y746;
    double x747, y747;
    double x748, y748;
    double x749, y749;
    double x750, y750;
    double x751, y751;
    double x752, y752;
    double x753, y753;
    double x754, y754;
    double x755, y755;
    double x756, y756;
    double x757, y757;
    double x758, y758;
    double x759, y759;
    double x760, y760;
    double x761, y761;
    double x762, y762;
    double x763, y763;
    double x764, y764;
    double x765, y765;
    double x766, y766;
    double x767, y767;
    double x768, y768;
    double x769, y769;
    double x770, y770;
    double x771, y771;
    double x772, y772;
    double x773, y773;
    double x774, y774;
    double x775, y775;
    double x776, y776;
    double x777, y777;
    double x778, y778;
    double x779, y779;
    double x780, y780;
    double x781, y781;
    double x782, y782;
    double x783, y783;
    double x784, y784;
    double x785, y785;
    double x786, y786;
    double x787, y787;
    double x788, y788;
    double x789, y789;
    double x790, y790;
    double x791, y791;
    double x792, y792;
    double x793, y793;
    double x794, y794;
    double x795, y795;
    double x796, y796;
    double x797, y797;
    double x798, y798;
    double x799, y799;
    double x800, y800;
    double x801, y801;
    double x802, y802;
    double x803, y803;
    double x804, y804;
    double x805, y805;
    double x806, y806;
    double x807, y807;
    double x808, y808;
    double x809, y809;
    double x810, y810;
    double x811, y811;
    double x812, y812;
    double x813, y813;
    double x814, y814;
    double x815, y815;
    double x816, y816;
    double x817, y817;
    double x818, y818;
    double x819, y819;
    double x820, y820;
    double x821, y821;
    double x822, y822;
    double x823, y823;
    double x824, y824;
    double x825, y825;
    double x826, y826;
    double x827, y827;
    double x828, y828;
    double x829, y829;
    double x830, y830;
    double x831, y831;
    double x832, y832;
    double x833, y833;
    double x834, y834;
    double x835, y835;
    double x836, y836;
    double x837, y837;
    double x838, y838;
    double x839, y839;
    double x840, y840;
    double x841, y841;
    double x842, y842;
    double x843, y843;
    double x844, y844;
    double x845, y845;
    double x846, y846;
    double x847, y847;
    double x848, y848;
    double x849, y849;
    double x850, y850;
    double x851, y851;
    double x852, y852;
    double x853, y853;
    double x854, y854;
    double x855, y855;
    double x856, y856;
    double x857, y857;
    double x858, y858;
    double x859, y859;
    double x860, y860;
    double x861, y861;
    double x862, y862;
    double x863, y863;
    double x864, y864;
    double x865, y865;
    double x866, y866;
    double x867, y867;
    double x868, y868;
    double x869, y869;
    double x870, y870;
    double x871, y871;
    double x872, y872;
    double x873, y873;
    double x874, y874;
    double x875, y875;
    double x876, y876;
    double x877, y877;
    double x878, y878;
    double x879, y879;
    double x880, y880;
    double x881, y881;
    double x882, y882;
    double x883, y883;
    double x884, y884;
    double x885, y885;
    double x886, y886;
    double x887, y887;
    double x888, y888;
    double x889, y889;
    double x890, y890;
    double x891, y891;
    double x892, y892;
    double x893, y893;
    double x894, y894;
    double x895, y895;
    double x896, y896;
    double x897, y897;
    double x898, y898;
    double x899, y899;
    double x900, y900;
    double x901, y901;
    double x902, y902;
    double x903, y903;
    double x904, y904;
    double x905, y905;
    double x906, y906;
    double x907, y907;
    double x908, y908;
    double x909, y909;
    double x910, y910;
    double x911, y
```



```

107 pidAlgorithm.request.lastError = 0;
108 pidAlgorithm.request.totalValue = 0;
109 pidAlgorithm.request.T = 10.0 / 1000.0; // convert time to seconds
110 bool firstPid = true;
111 bool turning = false;
112 // Quickly publish a zero movement twist in case the last program terminated
113 // with the robot moving
114 driver.publish(geometry_msgs::Twist());
115 ros::spinOnce();
116 rate.sleep();
117
118 while (ros::ok()) {
119     // Would love to extract all of these node client calls so they do not
120     // take up flow time much, but the continue function is to them...
121     if (sonarReader.call(sonarReadingSrv)) {
122         ROS_ERROR("Failed to use assignment1/sonar_wrapper: is it running?");
123         rate.sleep();
124         continue;
125     }
126     geometry_msgs::Twist movement;
127     if (sonarReadingSrv.response.readings.distance != UINT16_MAX) {
128         // Test if we get the same reading five times in a row
129         // it happens another five times in a row
130         if (turning) {
131             // publish zero movement
132             driver.publish(movement);
133             for (int i = 0; i < 5; i++) {
134                 // Test if we get the same reading five times
135                 rate.sleep();
136                 ros::spinOnce();
137                 if (sonarReader.call(sonarReadingSrv)) {
138                     ROS_ERROR("Failed to use assignment1/sonar_wrapper: is it running?");
139                     rate.sleep();
140                     continue;
141                 }
142                 if (sonarReadingSrv.response.readings.distance != UINT16_MAX) {
143                     continue;
144                 }
145                 // Enter turning mode
146                 turning = true;
147             }
148             // We can start driving forwards
149             if (turning) {
150                 // quickly publish a zero movement to stop moving
151                 turning = false;
152                 driver.publish(movement);
153                 rate.sleep();
154                 continue;
155             }
156         }
157         #ifdef NOISY_SONAR
158             // Just go ahead and generate the variance now
159             // Use a correct method because we use spin
160             if (sonar(kalmanFilterSrv.request.R_i)) {
161                 std::vector<uint16_t> sonarSamples;
162                 while (ros::ok() && variancesamples > 0) {
163                     variancesamples =
164                         if (variancesamples % 100 == 0) {
165                             ROS_INFO("collecting samples for variance calculation... N° PRIu32 * left", variancesamples);
166                         }
167                     ros::spinOnce();
168                     if (sonarReader.call(sonarReadingSrv)) {
169                         ROS_ERROR("Failed to use assignment1/sonar_wrapper: is it running?");
170                         rate.sleep();
171                         continue;
172                     }
173                     sonarSamples.push_back(sonarReadingSrv.response.readings.distance);
174                     rate.sleep();
175                 }
176                 kalmanFilterSrv.request.R_i = calculateVariance(sonarSamples);
177                 // We're effectively going to skip the first run by doing
178                 // v_i estimate = x_i estimate. We can just use the last
179                 // read output from reading the samples though.
180                 // x_i = x_i
181                 kalmanFilterSrv.request.x_i = sonarSamples[sonarSamples.size() - 1];
182                 kalmanFilterSrv.request.y_i estimate =
183                     kalmanFilterSrv.request.x_i;
184                 kalmanFilterSrv.request.P_i estimate =
185                     kalmanFilterSrv.request.R_i;
186                 if (ModelState.call(ModelStateSrv)) {
187                     ROS_ERROR("Failed to use gazebo/get_model_state: is it running?");
188                     rate.sleep();
189                     continue;
190                 }
191                 lastPose = ModelStateSrv.response.pose;
192             } else {
193                 // variance was not calculated, so the actual filter instead
194                 if (ModelState.call(ModelStateSrv)) {
195                     ROS_ERROR("Failed to use gazebo/get_model_state: is it running?");
196                     rate.sleep();
197                     continue;
198                 }
199                 // Sonars makes the units into centimetres instead so follow
200                 // units
201                 double && euclidean = euclideanDistance(lastPose, ModelStateSrv.response.pose) * 100;
202                 ROS_INFO("Euclidean distance is %f", euclidean);
203                 kalmanFilterSrv.request.y_i estimate =
204                     kalmanFilterSrv.request.y_i - euclidean;
205                 lastPose = ModelStateSrv.response.pose;
206             }
207             kalmanFilterSrv.request.x_i = sonarReadingSrv.response.readings.distance;
208             if (kalmanFilter.call(kalmanFilterSrv)) {
209                 ROS_ERROR("Failed to call kalman filter");
210                 rate.sleep();
211                 continue;
212             }
213             // Also make P_i estimate = P_i for the next run
214             ROS_INFO("P_i estimate (before) = %f, P_i = %f", kalmanFilterSrv.request.P_i estimate, kalmanFilterSrv.response.P_i);
215             kalmanFilterSrv.request.P_i estimate =
216                 kalmanFilterSrv.response.P_i;
217             ROS_INFO("P_i estimate (after) = %f", kalmanFilterSrv.request.P_i estimate);
218         }
219         #else
220             pidAlgorithm.request.error = kalmanFilterSrv.response.y_i;
221         #endif
222         pidAlgorithm.request.error = (double)sonarReadingSrv.response.readings.distance;
223         #endif
224         ROS_INFO("Read error as %f", pidAlgorithm.request.error);
225         ROS_INFO("Calling PID service with K_p = %f", pidAlgorithm.request.K_p);
226         if (PID_service.call(pidAlgorithmSrv)) {
227             ROS_ERROR("Failed to use assignment1/pid_algorithm: is it running?");
228             rate.sleep();
229             continue;
230         }
231         movement.linear.x = pidAlgorithmSrv.response.y;
232         // Reset (PID) requests based on the responses for the next run
233         pidAlgorithm.request.lastError =
234             pidAlgorithmSrv.request.error;
235         pidAlgorithm.request.totalValue =
236             pidAlgorithmSrv.response.totalValue;
237     }
238     driver.publish(movement);
239     Ros_INFO("Published linear = %f angular = %f",
240             movement.linear.x, movement.angular.z);
241     ros::spinOnce();
242     rate.sleep();
243 }
244 return EXIT_SUCCESS;
245 }

```

Figure 8: Implementation of the controller. Read as top left -&gt; top right -&gt; bottom left -&gt; bottom right

## 6 Compiling and replicating the results

The package should be extracted into a catkin workspace package and catkin\_make should be used. To launch the non-noisy version, launch sonars (from assignment1\_setup) assignment1\_pidAlgorithm, assignment1\_sonarWrapper and assignment1\_controller. To do the Kalman filter launch noisy\_sonars (from assignment1\_setup) assignment1\_pidAlgorithm, assignment1\_sonarWrapper, assignment1\_kalmanFilter, and assignment1\_kalman\_controller.

## References

- [Bhandari, 2020] Bhandari, P. (2020). Understanding and calculating variance. <https://www.scribbr.com/statistics/variance/>.
- [M., 2000] M., A. (2000). Pid control. <https://www.eolss.net/ebooks/Sample%20Chapters/C18/E6-43-03-03.pdf>.
- [Siegward et al., 2011] Siegward, Roland, Nourbakhsh, I. R., and Scaramuzza, D. (2011). Introduction to autonomous mobile robots second edition. [https://learn-ap-southeast-2-prod-fleet01-xythos.content.blackboardcdn.com/5bb70f08ac35e/8023721?X-Blackboard-Expiration=1619870400000&X-Blackboard-Signature=550omC3C9LgQebBieXZLrHODLw7EyXD7bB0n8vVfmOU%3D&X-Blackboard-Client-Id=101056&response-cache-control=private%2C%20max-age%3D21600&response-content-disposition=inline%3B%20filename%2A%3DUTF-8%27%27Introduction%2520to%2520Autonomous%2520Mobile%2520Robots\\_2nd.pdf&response-content-type=application%2Fpdf&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20210501T060000Z&X-Amz-SignedHeaders=host&X-Amz-Expires=21600&X-Amz-Credential=AKIAYDKQRRYZBCCQFY5%2F20210501%2Fap-southeast-2%2Fs3%2Faws4\\_request&X-Amz-Signature=a86bfb6cd33b718b655f848704fa7b055a939b711e9b60af19b85768bdc61ec5](https://learn-ap-southeast-2-prod-fleet01-xythos.content.blackboardcdn.com/5bb70f08ac35e/8023721?X-Blackboard-Expiration=1619870400000&X-Blackboard-Signature=550omC3C9LgQebBieXZLrHODLw7EyXD7bB0n8vVfmOU%3D&X-Blackboard-Client-Id=101056&response-cache-control=private%2C%20max-age%3D21600&response-content-disposition=inline%3B%20filename%2A%3DUTF-8%27%27Introduction%2520to%2520Autonomous%2520Mobile%2520Robots_2nd.pdf&response-content-type=application%2Fpdf&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20210501T060000Z&X-Amz-SignedHeaders=host&X-Amz-Expires=21600&X-Amz-Credential=AKIAYDKQRRYZBCCQFY5%2F20210501%2Fap-southeast-2%2Fs3%2Faws4_request&X-Amz-Signature=a86bfb6cd33b718b655f848704fa7b055a939b711e9b60af19b85768bdc61ec5).